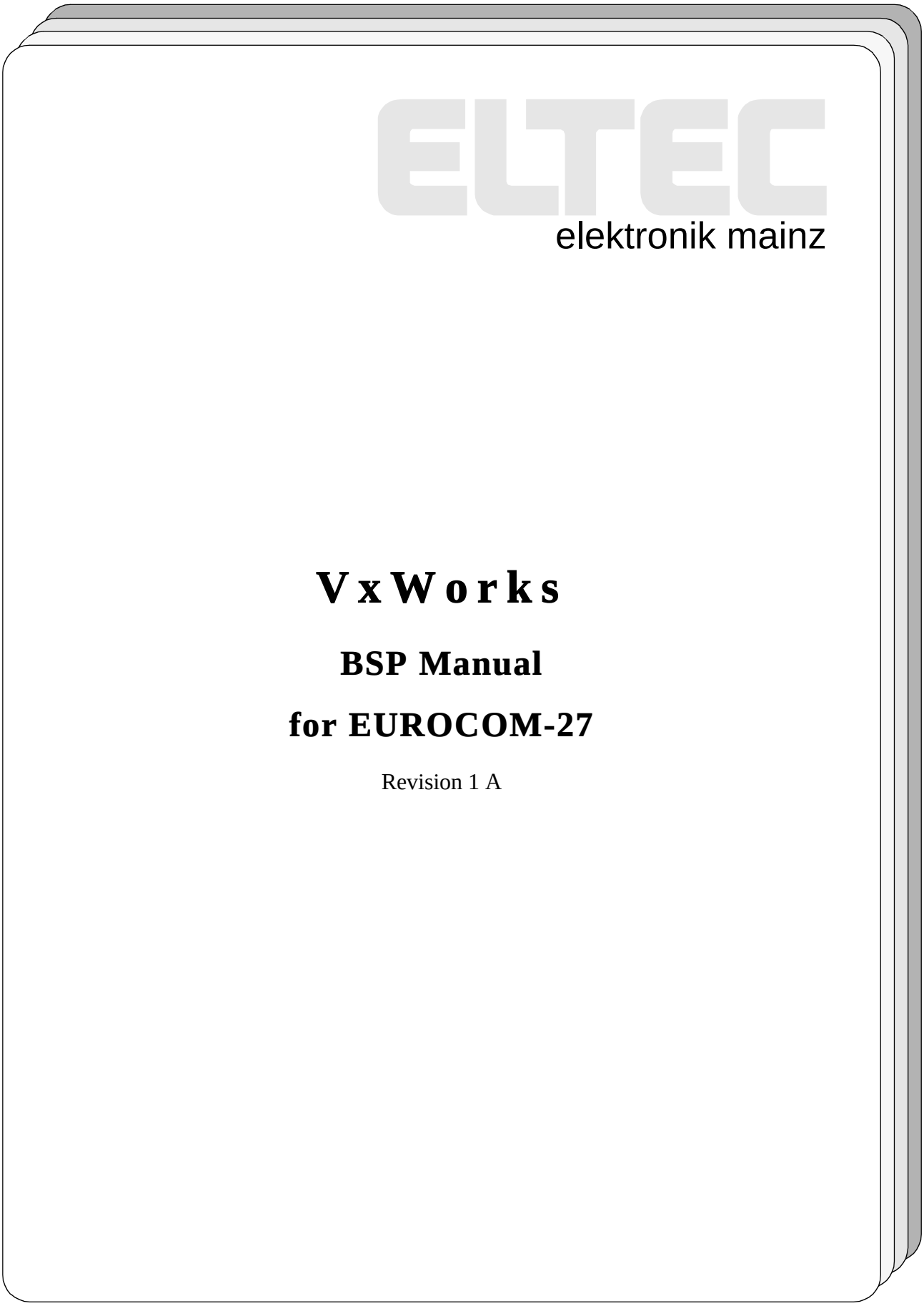

VxWorks

BSP Manual

for EUROCOM-27

Revision 1 A

Rev.	Changes	Date
1 A	First Edition valid for VxWorks 5.2	23.10.95, F.G.

DISCLAIMER!
The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. ELTEC reserves the right to make changes to any products to improve reliability, function or design. ELTEC does not assume any liability arising out of the application or use of any product or circuit described in this manual; neither does it convey any license under its patent rights nor the rights of others. ELTEC products are not authorized for use as components in life support devices or systems intended for surgical implant into the body or intended to support or sustain life. Buyer agrees to notify ELTEC of any such intended end use whereupon ELTEC shall determine availability and suitability of its product or products for the use intended.

ELTEC points out that there is no legal obligation to document internal relationships between any functional modules, realized in either hardware or software, of a delivered entity.

This document contains copyrighted information. All rights including those of translation, reprint, broadcasting, photomechanical or similar reproduction and storage or processing in computer systems, in whole or in part, are reserved.

EUROCOM is a trademark of ELTEC Elektronik AG. Other brands and their products are trademarks of their respective holders and should be noted as such.

© 1995 ELTEC Elektronik AG, Mainz

ELTEC Elektronik AG
Galileo-Galilei-Str. 11 Postfach 421363
D-55129 Mainz D-55071 Mainz

Telephone +49 (6131) 918-0
Telefax +49 (6131) 918-199

Table of Contents

Page

eltec27(T) 1
Introduction 1
Boot ROMs 1
Jumpers 1
Switches 2
Devices 3
Special Considerations 4
Board Layout 5
sysLib(1) 6
if_il(1) 10
ncr720Lib(1) 14
tyCoDrv(1) 17
fbLib(1) 18
cd2400CoDevCreate(2) 19
cd2400CoDrv(2) 20
eltecConDevCreate(2) 21
eltecConDrv(2) 22
fbDisplay(2) 23
fbFlush(2) 24
fbInitConsole(2) 25
fbSetColor(2) 26
ilattach(2) 27
ncr720CtrlCreate(2) 28
ncr720CtrlInit(2) 29
ncr720SetHwRegister(2) 30
ncr720Show(2) 31
sysAcfailConnect(2) 32
sysAuxClkConnect(2) 33
sysAuxClkDisable(2) 34
sysAuxClkEnable(2) 35
sysAuxClkRateGet(2) 36
sysAuxClkRateSet(2) 37
sysBspRev(2) 38
sysBusIntAck(2) 39
sysBusIntGen(2) 40
sysBusTas(2) 41
sysBusToLocalAdrs(2) 42
sysClkConnect(2) 43
sysClkDisable(2) 44

Page

sysClkEnable(2) 45
sysClkInt(2) 46
sysClkRateGet(2) 47
sysClkRateSet(2) 48
sysClockGet(2) 49
sysClockPrint(2) 50
sysClockSet(2) 51
sysClockStop(2) 52
sysFrontPanelSwitch(2) 53
sysHwInit(2) 54
sysIlaccIntEnable(2) 55
sysIntDisable(2) 56
sysIntEnable(2) 57
sysLocalToBusAdrs(2) 58
sysMailboxConnect(2) 59
sysMailboxEnable(2) 60
sysMemTop(2) 61
sysModel(2) 62
sysNvRamGet(2) 63
sysNvRamSet(2) 64
sysProcNumGet(2) 65
sysProcNumSet(2) 66
sysScsiInit(2) 67
sysScsiMode(2) 68
sysSetLed(2) 69
sysSysfailConnect(2) 70
sysToMonitor(2) 71
sysVicBlkAdj(2) 72
sysVicBlkCopy(2) 73
sysVicBlkEnable(2) 74
sysVicShow(2) 75
tyCoDevCreate(2) 76
tyCoDrv(2) 77

Technical Action Request Form Sheet
Reader Comments Form Sheet

Conventions

If not otherwise specified, addresses are written in hexadecimal notation and identified by a leading dollar sign ("$").

Signal names preceded by a slash ("/"), indicate that this signal is either active low or that this signal becomes active with the trailing edge.

b	bit
B	byte
K	kilo, means the factor 400 in hex (1024 decimal)
M	mega, the multiplication with 100 000 in hex (1 048576 decimal)
MHz	1 000 000 Hertz

eltec27(T)

Name ELTEC EUROCOM-27

Introduction The following provides board-specific information necessary to run VxWorks. Before running VxWorks, verify that the board runs in the factory configuration using vendor-supplied monitor program (RMon) and jumper settings, and check operation through the RS232 connection.

Boot ROMs The EUROCOM-27 uses a flash EPROM for the resident monitor program (RMon) which is always resident on the EUROCOM-27. The RMon performs various initialization functions on the board and then optionally transfers to the program resident within the user EPROM. The user EPROM, located at U1602, is used to program VxWorks. This EPROM must be a 32-pin device and can be from 128 KB to 1 MB (27C010 to 27C080).

The RMon needs to be set up such that the RMon automatically starts execution of the program resident in the user EPROM memory. To set up the automatic start of the program in user EPROM, set rotary switch number 2 to position 8 or higher. When the switch is set in this manner, the RMon will transfer control to the program located in the user EPROM.

The EUROCOM-27 has nonvolatile RAM; thus, boot parameters are preserved whenever the system is powered off.

To load VxWorks, and for more information, follow the instructions in the chapter Getting Started of the VxWorks Programmer's Guide.

The delivered Boot EPROM is designed for the most common board configuration.
For the EUROCOM-27 this corresponds to 8 MB RAM, graphics, SCSI and network devices on board. If the board differs from this combination, the standard Boot EPROM may hang up the system (e.g. less memory or SCSI chip missing). If the system hangs, config.h and/or eltec27.h must be modified and a new Boot EPROM must be rebuilt. Do not mix up different versions of Boot EPROM and BSP (e.g. Boot EPROM Version 5.11 - BSP Version 5.2).

Jumpers The EUROCOM-27 is factory configured for immediate operation with VxWorks. The table below lists the jumpers for the EUROCOM-27. Starred settings (*) indicate the factory default and are appropriate for use with VxWorks.

Jumper	Description	
J1201	Digclk inversion	(1-2 * not inverted) (2-3 inverted)
J1202	TTL Video Outputs	(open * when disabled) (closed when enabled)
J1401	Watchdog period	(closed * when 100 ms) (open when 1.6 seconds)
J1601	Flash programming voltage	(open * when not present) (closed when present)
J1605	Pin1 connection for EPROM	(1-2 * when 5 volts) (2-3 when A19)
J1702	EEPROM write enable	(closed * when disabled) (open when enabled)
J1801	Snooping of secondary computer	(1-2 * ignore) (2-3 observe)
J1703	VMEbus Interrupts for secondary CPU	

Switches

The EUROCOM-27 has two rotary switches on the front panel of the processor board. Switch number 1 is used to select the VMEbus address (A32 and A16) for RMon. The addressing is as defined in the following table.

Position	A32 Space	A16 Space
F	0xF0000000	0xF000
E	0xE0000000	0xE000
D	0xD0000000	0xD000
2	0x20000000	0x2000
1	0x10000000	0x1000
0	RMon defined	RMon defined

A24 addressing is always disabled by using the RMon defaults.

VxWorks can inherit the addressing assignments set up by RMon, if desired. When VxWorks is built with the USE_RMON_CONFIG defined, then VxWorks will use the RMon addressing. When USE_RMON_CONFIG is not defined, the addressing is set as per the LOCAL_MEM_BUS_ADRS definition within config.h and the processor number.

Switch number 2 is used to select the start up mode that is used by RMon. For automatic transfer to the VxWorks programmed in the user EPROM, this switch must be set to positions 8 - F. All other positions will not perform an automatic start of VxWorks. When the VxWorks kernel is built to include support for the selection of console port by switch, the switch specifies the port to use for the VxWorks console, as follows:

Position	VxWorks_Device
8	/tyCo/0
9	/tyCo/1
A	/tyCo/2
B	/tyCo/3
C	/tyCo/4

To enable this operation the SELECT_CONSOLE_SWITCH must be defined within the config.h file.

Switch S3 on the front panel is used to select VME system controller functions. When the EUROCOM-27 is placed in slot 1 of the VME chassis, the system controller option should be selected. When another processor is placed in slot 1, then that processor will usually fulfil the system controller function and the S3 should not select the system controller function.

Devices The eltecSerial.c Cirrus Logic CD2401 try driver is provided for the four on-board serial ports; see the manual entry for tyCoDrv. The eltecConsole.c driver provides support for the graphics/keyboard interface. The console driver will inherit the characteristics of the monitor as specified within the RMon monitor program. This includes the screen dimensions and background/foreground colors. When the graphics/ keyboard support is selected when the VxWorks kernel is built, the graphics/keyboard will be accessed as /tyCo/0 and the serial ports will be accessed as /tyCo/1 through /tyCo/4.
The INCLUDE_GRAPHICS_KEYBOARD is defined within the eltec27.h file to include the support for the console driver.

The chip drivers included are:

- mk48TxxClock.c - MK48Txx time keeper
- nvRam.c - nonvolatile RAM
- vic064Vme.c - Cypress VIC064 VME interface
- z8536Timer.c - Zilog Counter/Timer Parallel I/O (CIO) timer

A network interface for the on-board ILACC chip allows VxWorks to run without an additional network board. The interface is called "il" and should be specified as the boot device to the boot ROMs.

A device driver for the NCR53C720 SCSI Input Output Processor (SIOP) is provided. The INCLUDE_SCSI directive must be enabled in config.h.

Special Considerations

The ILACC chip needs to be informed of its Ethernet address before it can attach to the LAN. The address is obtained from the RMon EPROM which is unique for each board.

Console connection can be via any of the serial ports or the graphics/ keyboard interface. Selection of the console port is made through the VxWorks kernel build process. When enabled the console port can be selected via the setting of switch 2.

The NCR53C520 SCSI library supports SCSI wide and SCSI fast, however, these features are currently not supported by the SCSI library. Therefore, special functions are present to place a compatible SCSI device in the fast and/or wide mode. See the manual pages for sysScsiMode for additional information.

The EUROCOM-27 has a watchdog function that when enabled must be triggered at a minimum of the frequency selected by the J1401 setting (100 ms or 1.6 seconds). The INCLUDE_WDOG definition within config.h enables this feature.

The VIC064 VME interface driver provides an optional VME DMA copy function which is enabled by the inclusion of the INCLUDE_VIC_BLK_XFER definition within the config.h file. See the manual entry for sysVicBlkCopy for additional information on block copies.

The MK48Txx time keeper module on the EUROCOM-27 can be used to provide system time for VxWorks. If the definition INCLUDE_MK48TIME is defined in the config.h file, the time keeper functions are included. The ANSI time support also needs to be included within the VxWorks kernel which is the default. The definition INCLUDE_ANSI_TIME controls the inclusion of the ANSI time

functions. When the time keeper functions are included and the DOS file system is also included, the DOS files are stamped with the date and time as maintained by the MK48Txx.

Board Layout The EUROCOM-27 does not have any jumpers that need be in set positions for operation with VxWorks. Refer to the EUROCOM-27 technical manual for the locations of jumpers, switches, and EPROMs.

See Also Programmer's Guide: Getting Started,
Configuration RMon Monitor Program - Programmers Reference Manual,
RMon User Manual - Software Manual,
EUROCOM-27 Technical Manual

sysLib(1)

Name sysLib - ELTEC EUROCOM-27 system-dependent library

Synopsis

sysClockGet()	get the contents of the clock
sysClockSet()	sets the nonvolatile clock
sysClockStop()	stops the nonvolatile clock
sysClockPrint()	prints value stored in the nonvolatile clock
sysNvRamGet()	get the contents of nonvolatile RAM
sysNvRamSet()	write to nonvolatile RAM
sysIntDisable()	disable a bus interrupt level
sysIntEnable()	enable a bus interrupt level
sysBusIntAck()	acknowledge a bus interrupt
sysBusIntGen()	generate a bus interrupt
sysMailboxConnect()	connect a routine to the mailbox interrupt
sysMailboxEnable()	enable the mailbox interrupt
sysVicBlkEnable()	initialize the DMA copy operation for the VIC064.
sysVicBlkAdj()	adjust burst length and interleave period for DMA transfers
sysVicBlkCopy()	copy blocks over the VMEbus using the VIC 's DMA feature
sysVicShow()	displays the contents of the VIC064 registers
sysClkConnect()	connect a routine to the system clock interrupt
sysClkInt()	handle a system clock interrupt
sysClkDisable()	turn off system clock interrupts
sysClkEnable()	turn on system clock interrupts
sysClkRateGet()	get the system clock rate
sysClkRateSet()	set the system clock rate
sysAuxClkConnect()	connect a routine to the auxiliary clock interrupt
sysAuxClkDisable()	turn off auxiliary clock interrupts
sysAuxClkEnable()	turn on auxiliary clock interrupts
sysAuxClkRateGet()	get the auxiliary clock rate
sysAuxClkRateSet()	set the auxiliary clock rate
sysModel()	return the model name of the CPU board
sysBspRev()	return the bsp version with the revision e.g. 1.0/<x>
sysHwInit()	initialize the CPU board hardware
sysFrontPanelSwitch()	read the front panel switches
sysSetLed()	set value in LED
sysMemTop()	get the address of the top of memory
sysToMonitor()	transfer control to the ROM monitor
sysLocalToBusAdrs()	convert a local address to a bus address
sysBusToLocalAdrs()	convert a bus address to a local address

sysProcNumGet()	get the processor number
sysProcNumSet()	set the processor number
sysBusTas()	test and set a location across the bus
sysSysfailConnect()	connect a routine to the SYSFAIL signal
sysAcfailConnect()	connect a routine to the ACFAIL signal
sysIlaccIntEnable()	enable the ILACC interrupt level

```
time_t sysClockGet
    (struct tm *time)
STATUS sysClockSet
    (UINT8 date, UINT8 month, UINT8 year, UINT8 hour,
    UINT8 minute, UINT8 second)
STATUS sysClockStop
    (void)
STATUS sysClockPrint
    (void)
STATUS sysNvRamGet
    (char *string, int strLen, int offset)
STATUS sysNvRamSet
    (char *string, int strLen, int offset)
STATUS sysIntDisable
    (int intLevel)
STATUS sysIntEnable
    (int intLevel)
int sysBusIntAck
    (int intLevel)
STATUS sysBusIntGen
    (int level, int vector)
STATUS sysMailboxConnect
    (FUNCPTR routine, int arg)
STATUS sysMailboxEnable
    (char *mailboxAdrs)
STATUS sysVicBlkEnable
    (VOIDFUNCPTR * vector, int level, int burstLength,
    int interleave)
STATUS sysVicBlkAdj
    (int burstLength, int interleave)
STATUS sysVicBlkCopy
    (char * localAddr, char * remoteAddr, int nbytes,
    BOOL toVme, BOOL doD64)
intsysVicShow()
STATUS sysClkConnect
    (FUNCPTR routine, int arg)
void sysClkInt
    (void)
void sysClkDisable
    (void)
```

```
void sysClkEnable
    (void)
int sysClkRateGet
    (void)
STATUS sysClkRateSet
    (int ticksPerSecond)
STATUS sysAuxClkConnect
    (FUNCPTR routine, int arg)
void sysAuxClkDisable
    (void)
void sysAuxClkEnable
    (void)
int sysAuxClkRateGet
    (void)
STATUS sysAuxClkRateSet
    (int ticksPerSecond)
char *sysModel
    (void)
char * sysBspRev
    (void)
void sysHwInit
    (void)
int sysFrontPanelSwitch
    (int swNum)
int sysSetLed
    (int hexVal)
char *sysMemTop
    (void)
STATUS sysToMonitor
    (int startType)
STATUS sysLocalToBusAdrs
    (int adrsSpace, char *localAdrs, char **pBusAdrs)
STATUS sysBusToLocalAdrs
    (int adrsSpace, char *busAdrs, char **pLocalAdrs)
int sysProcNumGet
    (void)
void sysProcNumSet
    (int procNum)
BOOL sysBusTas
    (char *adrs)
STATUS sysSysfailConnect
    (VOIDFUNCPTR routine, int arg)
STATUS sysAcfailConnect
    (VOIDFUNCPTR routine, int arg)
int sysIlaccIntEnable
    (int level)
```

Description This library provides board-specific routines. The chip drivers included are:

- eltecSerial.c Cirrus CD2400 serial device module library
- mk48TxxClock.c MK48Txx time keeper clock module library
- nvRam.c nonvolatile RAM library
- vic064Vme.c Cypress VIC068 VMEbus interface controller library
- z8536Timer.c Zilog 8536 Counter/Timer device library

Include Files sysLib.h

See Also Programmer's Guide: Configuration

if_il(1)

Name if_il - AMD 79900 ILACC Ethernet network interface driver

Synopsis ilattach() - publish the interface, and initialize the driver and device

```
STATUS ilattach
    (int unit, char *devAdrs, int ivec, int ilevel,
    char *memAdrs, ULONG memSize, int memWidth, int spare,
    int spare2)
```

Description
General Information

This module implements the AMD 79900 ILACC Ethernet network interface driver.

This driver is designed to achieve a moderate level of genericism, across the range of architectures and targets that VxWorks 5.1 supports. This allows the driver to be run, unmodified, on a variety of target hardware. To achieve this goal, the driver must be given several target-specific parameters, and some external support routines must be provided. These parameters, and the mechanisms used to communicate them to the driver, are detailed below. If any of the assumptions mentioned below are not true for your particular hardware, this driver will probably not function correctly. This driver supports only one ILACC unit per CPU. The driver can be configured to support big-endian or little-endian architectures.

Board Layout This device is on-board. No jumpering diagram required.

External Interface This driver provides the standard external interface with the following exceptions. All initialization is performed within the attach() routine and there is no separate init() routine. Therefore, in the global interface structure, the function pointer to the init() routine is NULL.

There is one user-callable routine: ilattach(). See the manual entry for this routine for usage details.

Target Specific Parameters

- bus mode

  This parameter is globally accessed.

  The ILACC control register #3 determines the bus mode of the device. This allows the support of big-endian and little-endian architectures. This parameter, defined as "u_short ilCSR_3B", is the value that will be placed into ILACC control register #3. The default value supports Motorola type busses. See the ILACC manual to change this parameter.

- base address of device registers
  This parameter is passed to the driver with the ilattach() routine.

  The ILACC presents two registers to the external interface, the RDP register, and the RAP register. This driver assumes that these two registers occupy two unique addresses in a memory space that is directly accessible by the CPU executing this driver. This driver assumes that the RDP register is mapped at a lower address than the RAP register, and is therefore considered the "base address".

  This parameter indicates to the driver where to find the RDP register.

- interrupt vector
  This parameter is passed to the driver with the ilattach() routine.

  This driver configures the ILACC device to generate hardware interrupts for various events within the device. Therefore, this driver contains an interrupt handler routine. This driver will call the VxWorks system function intConnect() to connect its interrupt handler to the interrupt vector generated as a result of the ILACC interrupt.

- interrupt level
  This parameter is passed to the driver with the ilattach() routine.

  Some target hardware use additional interrupt controller devices to help organize and service the various interrupt sources. This driver avoids all board-specific knowledge of such devices. During the initialization of this driver, an external routine is called to perform any board-specific operations required to allow the servicing of an ILACC interrupt. This routine is described below. This parameter is passed to the external routine.

- shared memory address
  This parameter is passed to the driver with the ilattach() routine.

  The ILACC device is a DMA type of device and typically shares access to some region of memory with the CPU. This driver is designed for systems that directly share memory between the CPU and the ILACC. This driver assumes that this shared memory is directly available to this driver without any arbitration or timing concerns.

  This parameter may be used to specify an explicit memory region for use by the ILACC. This should be used on hardware that restricts the ILACC to a particular memory region. The constant NONE may be used to indicate that there are no memory limitations. In this case, the driver will attempt to allocate the shared memory from the system space.

- shared memory size
  This parameter is passed to the driver with the ilattach() routine.

  This parameter can be used to explicitly limit the amount of shared memory (bytes) this driver will use. The constant NONE may be used to indicate no specific size limitation. This parameter is only used if a specific memory region is being provided to the driver.

- shared memory width
  This parameter is passed to the driver with the ilattach() routine.

  Some target hardware that restricts the shared memory region to a specific location, also restricts the access width to this region by the CPU. On these targets, performing an access of an invalid width will cause a bus error.

  This parameter can be used to specify the number of bytes of access width to be used by the driver during access to the shared memory. The constant NONE may be used to indicate no restrictions.

  The current internal implementation of supporting this mechanism is not robust, and may not work on all targets requiring these restrictions.

- shared memory buffer size
  This parameter is passed to the driver with the ilattach() routine.

  The driver and ILACC device exchange network data in buffers. This parameter allows you to limit the size of these individual buffers. A value of zero indicates that the default buffer size should be used. The default buffer size is large enough to hold a maximum size Ethernet packet.

  Use of this parameter should be extremely rare. Network performance will be affected, since the target will no longer be able to receive all legal sizes of packets.

- Ethernet address
  This parameter is obtained directly from a global memory location.

  During initialization, the driver needs to know the Ethernet address for the ILACC device. The driver assumes that this address is available in a global, 6 byte, character array, named ilEnetAddr[]. This array is typically created and stuffed by the BSP code.

External Support Requirements

This driver requires one external support routine to be provided.

- void sysIlaccIntEnable (int level)
  This routine provides a target-specific enable of the interrupt for the ILACC device. This typically involves interrupt controller hardware, either internal or external to the CPU. This routine is called once from the ilattach() routine.

System Resource Usage

The driver requires the following system resources:

- one mutual exclusion semaphore
- one interrupt vector
- 5016 bytes in code section (text)
- 28 bytes in the initialized data section (data)
- 2244 bytes in the uninitialized data section (bss)

The sections are quoted for the MC68040 architecture and will vary for other architectures.

If the driver allocates the memory to share with the ILACC, it does so by calling the cacheDmaMalloc() routine. The size requested is 80,542 bytes. If a memory region is provided to the driver, the size of this region is adjustable to suit your needs.

The ILACC can only be operated if the shared memory region is write-coherent with regards to the data cache. The driver cannot maintain cache coherency for the device for data that is written by the driver. This is because fields within the shared structures are asynchronously modified by both the driver, and the device, and these fields may share the same cache line.

See Also ifLib

ncr720Lib(1)

Name ncr720Lib - NCR53C720 SCSI I/O Processor (SIOP) library

Synopsis

ncr720CtrlCreate()	- create a control structure for the SIOP
ncr720CtrlInit()	- initialize a control structure for the SIOP
ncr720SetHwRegister()	- set hardware dependent registers
ncr720Show()	- display values of all readable ncr720 (SIOP) registers

```
NCR_720_SCSI_CTRL *ncr720CtrlCreate
    (UINT8 *baseAdrs, UINT freqValue)
STATUS ncr720CtrlInit
    (NCR_720_SCSI_CTRL *pSiop, int scsiCtrlBusId,
    int scsiPriority)
STATUS ncr720SetHwRegister
    (SIOP *pSiop, NCR720_HW_REGS *pHwRegs)
STATUS ncr720Show
    (SCSI_CTRL *pScsiCtrl)
```

Description This is the I/O driver for the NCR53C720 SCSI I/O Processor (SIOP). It is designed to work in conjunction with scsiLib. This driver runs in conjunction with a script program for the NCR53C720 chip. The script uses the NCR53C720 DMA function for data transfers. This driver supports cache functions through cacheLib.

User Callable Routines Most of the routines in this driver are accessible only through the I/O system. Three routines, however, must be called directly: ncr720CtrlCreate() to create a controller structure, and ncr720CtrlInit() to initialize it. The NCR53C720 hardware registers need to be configured according to the hardware implementation. If the default configuration is not proper, the routine ncr720SetHwRegister() should be used to properly configure the registers.

Include Files ncr720.h, ncr720Script.h

See Also scsiLib Programmer ’s Guide: I/O System

cd2400CoDevCreate() (2)	create a device for an on-board serial port
cd2400CoDrv() (2)	tty driver initialization routine
eltecConDevCreate() (2)	create a device for the keyboard and graphics ports
eltecConDrv() (2)	console driver initialization routine
fbDisplay() (2)	display characters to the frame buffer
fbFlush() (2)	sets the entire screen to the specified color.
fbInitConsole() (2)	initialize the frame buffer for use with the VxWorks console
fbLib (1)	Frame Buffer console library
fbSetColor() (2)	sets the foreground and background colors
if_il (1)	AMD 79900 ILACC Ethernet network interface driver
ilattach() (2)	publish the interface, and initialize the driver and device
ncr720CtrlCreate() (2)	create a control structure for the SIOP
ncr720CtrlInit() (2)	initialize a control structure for the SIOP
ncr720Lib (1)	NCR53C720 SCSI I/O Processor (SIOP) library
ncr720SetHwRegister() (2)	set hardware dependent registers
ncr720Show() (2)	display values of all readable ncr720 (SIOP) registers
sysAcfailConnect() (2)	connect a routine to the ACFAIL signal
sysAuxClkConnect() (2)	connect a routine to the auxiliary clock interrupt
sysAuxClkDisable() (2)	turn off auxiliary clock interrupts
sysAuxClkEnable() (2)	turn on auxiliary clock interrupts
sysAuxClkRateGet() (2)	get the auxiliary clock rate
sysAuxClkRateSet() (2)	set the auxiliary clock rate
sysBspRev() (2)	return the bsp version with the revision e.g. 1.0/<x>
sysBusIntAck() (2)	acknowledge a bus interrupt
sysBusIntGen() (2)	generate a bus interrupt
sysBusTas() (2)	test and set a location across the bus
sysBusToLocalAdrs() (2)	convert a bus address to a local address
sysClkConnect() (2)	connect a routine to the system clock interrupt
sysClkDisable() (2)	turn off system clock interrupts
sysClkEnable() (2)	turn on system clock interrupts
sysClkInt() (2)	handle a system clock interrupt
sysClkRateGet() (2)	get the system clock rate
sysClkRateSet() (2)	set the system clock rate
sysClockGet() (2)	get the contents of the clock
sysClockPrint() (2)	prints value stored in the nonvolatile clock
sysClockSet() (2)	sets the nonvolatile clock

sysClockStop() (2)	stops the nonvolatile clock
sysFrontPanelSwitch() (2)	read the front panel switches
sysHwInit() (2)	initialize the CPU board hardware
sysIlaccIntEnable() (2)	enable the ILACC interrupt level
sysIntDisable() (2)	disable a bus interrupt level
sysIntEnable() (2)	enable a bus interrupt level
sysLib (1)	ELTEC EUROCOM-27 system-dependent library
sysLocalToBusAdrs() (2)	convert a local address to a bus address
sysMailboxConnect() (2)	connect a routine to the mailbox interrupt
sysMailboxEnable() (2)	enable the mailbox interrupt
sysMemTop() (2)	get the address of the top of memory
sysModel() (2)	return the model name of the CPU board
sysNvRamGet() (2)	get the contents of nonvolatile RAM
sysNvRamSet() (2)	write to nonvolatile RAM
sysProcNumGet() (2)	get the processor number
sysProcNumSet() (2)	set the processor number
sysScsiInit() (2)	initialize NCR720 SCSI chip
sysScsiMode() (2)	change the SCSI mode for a target
sysSetLed() (2)	set value in LED
sysSysfailConnect() (2)	connect a routine to the SYSFAIL signal
sysToMonitor() (2)	transfer control to the ROM monitor
sysVicBlkAdj() (2)	adjust burst length and interleave period for DMA transfers
sysVicBlkCopy() (2)	copy blocks over the VMEbus using the VIC 's DMA feature
sysVicBlkEnable() (2)	initialize the DMA copy operation for the VIC064
sysVicShow() (2)	displays the contents of the VIC064 registers
tyCoDevCreate() (2)	create a device for an on-board serial port
tyCoDrv (1)	ELTEC EUROCOM-27 tty driver
tyCoDrv() (2)	initialize the tty driver

tyCoDrv(1)

Name tyCoDrv - ELTEC EUROCOM-27 tty driver

Synopsis

cd2400CoDrv()	- tty driver initialization routine
cd2400CoDevCreate()	- create a device for an on-board serial port
tyCoDrv()	- initialize the tty driver
tyCoDevCreate()	- create a device for an on-board serial port

```
STATUS cd2400CoDrv
    (void)
STATUS cd2400CoDevCreate
    (char * name, int channel, int rdBufSize,
    int wrtBufSize)
STATUS tyCoDrv
    (void)
STATUS tyCoDevCreate
    (char *name, int channel, int rdBufSize, int wrtBufSize)
```

Description This is the driver for the ELTEC tty driver. It provides support for the Cirrus Logic CD2400 MPCC and the graphics/keyboard interfaces. The graphics/keyboard interface is treated as /tyCo/0 and the CD2400 ports start at /tyCo/1. The driver uses the CD2400 only in asynchronous mode.

User-callable Routines Most of the routines in this driver are accessible only through the I/O system. Two routines, however, must be called directly: tyCoDrv() to initialize the driver, and tyCoDevCreate() to create devices.
Before the driver can be used, it must be initialized by calling tyCoDrv(). This routine should be called exactly once, before any reads, writes, or calls to tyCoDevCreate(). Normally, it is called from usrRoot() in usrConfig.c. Before a terminal can be used, it must be created using tyCoDevCreate(). Each port to be used should have exactly one device associated with it by calling this routine.

Ioctl Functions This CD2401 driver responds to the same ioctl() codes as a normal tty driver; for more information, see the manual entry for tyLib. The available baud rates are: 50, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200, and 38400. In addition, the character size and parity is selected using the FIOBAUDRATE ioctl. The argument to the ioctl call, specifies the baud rate, character size, and parity. Bits 0-15 specify the baud rate, bits 26 and 27 specify the parity (0 = none, 1 = odd, and 2 = even), and bits 28 - 31 specify the character size (size of zero is use the default). The graphics/keyboard device responds to the same ioctl() codes as a normal tty driver.

See Also tyLib

fbLib(1)

Name fbLib - Frame Buffer console library

Synopsis

fbInitConsole()	- initialize the frame buffer for use with the VxWorks console
fbSetColor()	- sets the foreground and background colors
fbFlush()	- sets the entire screen to the specified color.
fbDisplay()	- display characters to the frame buffer

```
STATUS fbInitConsole
    (CON_DEV *dev, int baseAdrs, int height, int width)
VOID fbSetColor
    (CON_DEV *dev, char bgColor, char fgColor)
VOID fbFlush
    (CON_DEV *dev, char color)
VOID fbDisplay
    (CON_DEV *dev, char outChar)
```

cd2400CoDevCreate(2)

Name cd2400CoDevCreate() - create a device for an on-board serial port

Synopsis

```
STATUS cd2400CoDevCreate
    (
    char *  name,       /* name to use for this device      */
    int     channel,    /* physical channel for this device */
    int     rdBufSize,  /* read buffer size, in bytes       */
    int     wrtBufSize  /* write buffer size, in bytes      */
    )
```

Description This routine creates a device on a specified serial port. Each port to be used should have exactly one device associated with it by calling this routine.

For instance, to create the device "/tyCo/0", with buffer sizes of 512 bytes, the proper call would be:

```
tyCoDevCreate ("/tyCo/0", 0, 512, 512);
```

Returns OK, or ERROR if the driver is not installed or the channel is invalid.

See Also tyCoDrv

cd2400CoDrv(2)

Name cd2400CoDrv() - tty driver initialization routine

Synopsis
```
STATUS cd2400CoDrv (void)
```

Description This routine initializes the serial driver, sets up interrupt vectors, and performs hardware initialization of the serial ports.

This routine should be called exactly once, before any reads, writes, or calls to tyCoDevCreate(). Normally, it is called by usrRoot() in usrConfig.c.

Returns OK, or ERROR if the driver cannot be installed.

See Also tyCoDrv

eltecConDevCreate(2)

Name eltecConDevCreate() - create a device for the keyboard and graphics ports

Synopsis

```
STATUS eltecConDevCreate
    (
    char *  name,        /* Name to use for this device  */
    int     rdBufSize,   /* Read buffer size, in bytes   */
    int     wrtBufSize,  /* Write buffer size in bytes   */
    int     fbBaseAdrs,  /* base address of frame buffer */
    UINT16  width,       /* width of screen              */
    UINT16  height,      /* height of screen             */
    UINT8   fgColor,     /* background color             */
    UINT8   bgColor      /* foreground color             */
    )
```

Description This routine creates a device for keyboard and graphics ports.

Returns OK or ERROR if no driver or already exists.

See Also tyCoDrv

eltecConDrv(2)

Name eltecConDrv() - console driver initialization routine

Synopsis
```
STATUS eltecConDrv (void)
```

Description This routine initializes the console driver, sets up interrupt vectors, and performs hardware initialization of the keyboard and display.

Returns OK or ERROR if unable to install driver.

See Also tyCoDrv

fbDisplay(2)

Name fbDisplay() - display characters to the frame buffer

Synopsis

```
VOID fbDisplay
    (
    CON_DEV  *dev,    /* device descriptor    */
    char     outChar  /* character to display */
    )
```

Description This routine converts the requested characters to the required pixels within the frame buffer.

Returns N/A

See Also fbLib

fbFlush(2)

Name fbFlush() - sets the entire screen to the specified color.

Synopsis

```
VOID fbFlush
    (
    CON_DEV  *dev,  /* device descriptor */
    char     color  /* color             */
    )
```

Description This routine sets the entire screen to the specified color.

Returns N/A

See Also fbLib

fbInitConsole(2)

Name fbInitConsole() - initialize the frame buffer for use with the VxWorks console

Synopsis

```
STATUS fbInitConsole
    (
    CON_DEV  *dev,      /* device descriptor            */
    int      baseAdrs,  /* base address of frame buffer */
    int      height,    /* height of screen             */
    int      width      /* width of screen              */
    )
```

Description This routine initializes the frame buffer console library.

Returns OK or ERROR if unable to initialize the frame buffer.

See Also fbLib

fbSetColor(2)

Name fbSetColor() - sets the foreground and background colors

Synopsis

```
VOID fbSetColor
    (
    CON_DEV  *dev,     /* device descriptor */
    char     bgColor,  /* background color  */
    char     fgColor   /* foreground color  */
    )
```

Description This routine sets the foreground and background colors for the display.

Returns N/A

See Also fbLib

ilattach(2)

Name ilattach() - publish the interface, and initialize the driver and device

Synopsis

```
STATUS ilattach
    (
    int    unit,       /* unit number */
    char   *devAdrs,   /* ILACC i/o address */
    int    ivec,       /* interrupt vector */
    int    ilevel,     /* interrupt level */
    char   *memAdrs,   /* address of memory pool
                          (-1 == malloc it) */
    ULONG  memSize,    /* only used if memory pool is NOT
                          malloced */
    int    memWidth,   /* byte-width of data
                          (-1 == any width) */
    int    spare,      /* not used */
    int    spare2      /* not used */
    )
```

Description The routine publishes the "il" interface by filling in a network interface record and adding this record to the system list. This routine also initializes the driver and the device to the operational state.

The <memAdrs> parameter can be used to specify the location of the memory that will be shared between the driver and the device. The value NONE may be used to indicate that the driver should obtain the memory.

The <memSize> parameter is only valid if the <memAdrs> parameter is not set to NONE. In this case, this parameter indicates the size of the provided memory region.

The <memWidth> parameter sets the memory pool's data port width (in bytes); if NONE, any data width will be used.

Returns OK or ERROR.

See Also if_il

ncr720CtrlCreate(2)

Name ncr720CtrlCreate() - create a control structure for the SIOP

Synopsis

```
NCR_720_SCSI_CTRL *ncr720CtrlCreate
    (
    UINT8  *baseAdrs,  /* base address of the SIOP */
    UINT   freqValue   /* clock controller         */
    )
```

Description This routine creates an SIOP data structure and must be called before using an SIOP chip. It should be called only once for a given SIOP. Since it allocates memory for a structure needed by all routines in ncr720Lib, it must be called before any other routines in the library. After calling this routine, at least one call to ncr720CtrlInit() should be performed before any SCSI transactions are initiated using the SIOP.

A detailed description of the input parameters follows:

<baseAdrs> The address at which the CPU would access the lowest register of the SIOP.

<freqValue> The value at the SIOP SCSI CLK input. This is used to determine the clock period for the scsi core of the chip and the synchronous divider value for synchronous transfer. It is important to have the right timing on the SCSI bus.

Returns A pointer to NCR_720_SCSI_CTRL structure, or NULL if memory is unavailable or there are bad parameters.

See Also ncr720Lib

ncr720CtrlInit(2)

Name ncr720CtrlInit() - initialize a control structure for the SIOP

Synopsis

```
STATUS ncr720CtrlInit
    (
    NCR_720_SCSI_CTRL  *pSiop,        /* ptr to SIOP struct */
    int                scsiCtrlBusId, /* SCSI bus ID of this
                                         SIOP */
    int                scsiPriority   /* priority of a task
                                         when doing SCSI I/O */
    )
```

Description This routine initializes an SIOP structure, after the structure is created with ncr720CtrlCreate(). This structure must be initialized before the SIOP can be used. It may be called more than once if needed; however, it should only be called while there is no activity on the SCSI interface. Before using the SIOP, it must be initialized by calling this routine.

Before returning, this routine pulses RST (reset) on the SCSI bus, thus resetting all attached devices.

A detailed description of the input parameters follows:

<pSiop> a pointer to the NCR_720_SCSI_CTRL structure created with ncr720CtrlCreate().

<scsiCtrlBusId> the SCSI bus ID of the SIOP. Its value is somewhat arbitrary: seven (7), or highest priority, is conventional. The value must be in the range 0 - 7.

<scsiPriority> the priority to which a task is set when performing a SCSI transaction. Legal priorities are 0 to 255. Alternately, a -1 indicates that the priority should not be altered during SCSI transactions.

Returns OK, or ERROR if parameters are out of range.

See Also ncr720Lib

ncr720SetHwRegister(2)

Name ncr720SetHwRegister() - set hardware dependent registers

Synopsis

```
STATUS ncr720SetHwRegister
    (
    SIOP            *pSiop,   /* Pointer to SIOP infos */
    NCR720_HW_REGS  *pHwRegs  /* Pointer to a NCR720_HW_REGS
                                 info */
    )
```

Description This routine is used to set up the registers involved in the hardware implementation of the chip. Typically, this routine is called by the sysScsiInit() routine from the bsp. The input parameters are the pointer returned by ncr720CtrlCreate() and a pointer to a NCR720_HW_REGS structure that is filled with the logical values 0 or 1 for each bit of each register describe below. This routine includes only the bit registers that could modify the behavior of the chip. The default configuration used during ncr720CtlrCreate(), and ncr720CrtlInit() is
{0,0,0,0,1,0,0,0,0,0,0,0,1,0}

```
typedef struct
    {
    int ctest4Bit7;    /* Host bus multiplex mode */
    int ctest0Bit7;    /* Disable/enable burst cache capability */
    int ctest0Bit6;    /* Snoop control bit1 */
    int ctest0Bit5;    /* Snoop control bit0 */
    int ctest0Bit1;    /* invert tt1 pin (sync bus host mode only)*/
    int ctest2Bit5;    /* enable differential scsi bus capability*/
    int ctest3Bit0;    /* Set snoop pins mode */
    int dmodeBit7;     /* Burst Length transfer bit 1 */
    int dmodeBit6;     /* Burst Length transfer bit 0 */
    int dmodeBit5;     /* Function code bit FC2 */
    int dmodeBit4;     /* Function code bit FC1 */
    int dmodeBit3;     /* Program data bit (FC0) */
    int dmodeBit1;     /* user  programmable transfer type */
    int dcntlBit5;     /* Enable Ack pin */
    int dcntlBit1;     /* Enable fast arbitration on host port */
    } NCR720_HW_REGS;
```

To get a more detailed explanation regarding the description of each register involved see the User ’s Manual of the NCR53C720.

Note Because this routine writes to the chip registers you can't use it if there is any SCSI bus activity.

Returns OK or ERROR if any input parameter is NULL.

See Also ncr720Lib, ncr720.h, ncr720CtlrCreate()

ncr720Show(2)

Name ncr720Show() - Display values of all readable ncr720 (SIOP) registers

Synopsis

```
STATUS ncr720Show
    (
    SCSI_CTRL  *pScsiCtrl  /* ptr to SCSI controller info */
    )
```

Description Displays the state of the SIOP registers in a user-friendly way. Primarily used during debugging. The input parameter is the pointer to the SIOP info structure returned by the ncr720CtrlCreate() call.

Note The only readable register during a script execution is Istat register. If you use this routine during the execution of a SCSI command the result will be unpredictable.

Example

```
-> ncr720Show
NCR720 Registers
----------------
0xfff47000: Scntl3  = 0xa5 Scntl2  = 0x00 Scntl1  = 0x00 Scntl0  = 0x04
0xfff47004: Gpreg   = 0x00 Sdid    = 0x00 Sxfer   = 0x80 Scid    = 0x80
0xfff47008: Sbcl    = 0x00 Ssid    = 0x00 Socl    = 0x00 Sfbr    = 0x00
0xfff4700c: Sstat2  = 0x00 Sstat1  = 0x00 Sstat0  = 0x00 Dstat   = 0x80
0xfff47010: Dsa     = 0x00000000
0xfff47014: Istat   = 0x00
0xfff47018: Ctest3  = 0x00 Ctest2  = 0x21 Ctest1  = 0xf0 Ctest0  = 0x00
0xfff4701c: Temp    = 0x00000000
0xfff47020: Ctest6  = 0x00 Ctest5  = 0x00 Ctest4  = 0x00 Dfifo   = 0x00
0xfff47024: Dcmd/Ddc= 0x50000000
0xfff47028: Dnad    = 0x00066144
0xfff4702c: Dsp     = 0x00066144
0xfff47030: Dsps    = 0x00066174
0xfff47034: Scratch3= 0x00 Scratch2= 0x00 Scratch1= 0x00 Scratch0= 0x0a
0xfff47038: Dcntl   = 0x21 Dwt     = 0x00 Dien    = 0x37 Dmode   = 0x01
0xfff4703c: Adder   = 0x000cc2b8
0xfff47040: Sist1   = 0x00 Sist1   = 0x00 Sien1   = 0x00 Sien0   = 0x00
0xfff47046: Gpcntl  = 0x00 Macntl  = 0x00 Swide   = 0x00 Slpar   = 0x00
0xfff4704a: RespID  =                     Stime1  = 0x00 Stime0  = 0x00
0xfff4704c: Stest3  = 0x00 Stest2  = 0x00 Stest1  = 0x00 Stest0  = 0x00
0xfff47052: Sidl    = 0x0000
0xfff47056: Sodl    = 0x0000
0xfff4705a: Ssbdl   = 0x0000
0xfff4705c: ScracthB= 0x00000000
value = 0 = 0x0
```

See Also ncr720Lib, ncr720CtrlCreate()

Returns OK, or ERROR if <pScsiCtrl> and <pSysScsiCtrl> are both NULL.

sysAcfailConnect(2)

Name sysAcfailConnect() - connect a routine to the ACFAIL signal

Synopsis

```
STATUS sysAcfailConnect
    (
    VOIDFUNCPTR  routine,  /* routine to be connected to
                              ACFAIL signal */
    int          arg       /* argument with which to call
                              routine */
    )
```

Description This routine connects a specified routine to the board's ACFAIL signal.

Returns OK, or ERROR if the routine cannot be connected to the interrupt.

See Also sysLib, intConnect()

sysAuxClkConnect(2)

Name sysAuxClkConnect() - connect a routine to the auxiliary clock interrupt

Synopsis

```
STATUS sysAuxClkConnect
    (
    FUNCPTR  routine, /* routine called at each aux clock
                         interrupt */
    int      arg      /* argument with which to call
                         routine */
    )
```

Description This routine specifies the interrupt service routine to be called at each auxiliary clock interrupt. It also connects the clock error interrupt service routine.

Returns OK, or ERROR if the routine cannot be connected to the interrupt.

See Also sysLib, intConnect(), sysAuxClkEnable()

sysAuxClkDisable(2)

Name sysAuxClkDisable() - turn off auxiliary clock interrupts

Synopsis `void sysAuxClkDisable (void)`

Description This routine disables auxiliary clock interrupts.

Returns N/A

See Also sysLib, sysAuxClkEnable()

sysAuxClkEnable(2)

Name sysAuxClkEnable() - turn on auxiliary clock interrupts

Synopsis `void sysAuxClkEnable (void)`

Description This routine enables auxiliary clock interrupts.

Returns N/A

See Also sysLib, sysAuxClkDisable()

sysAuxClkRateGet(2)

Name sysAuxClkRateGet() - get the auxiliary clock rate

Synopsis `int sysAuxClkRateGet (void)`

Description This routine returns the interrupt rate of the auxiliary clock.

Returns The number of ticks per second of the auxiliary clock.

See Also sysLib, sysAuxClkEnable(), sysAuxClkRateSet()

sysAuxClkRateSet(2)

Name sysAuxClkRateSet() - set the auxiliary clock rate

Synopsis

```
STATUS sysAuxClkRateSet
    (
    int  ticksPerSecond  /* number of clock interrupts per
                            second */
    )
```

Description This routine sets the interrupt rate of the auxiliary clock. If the auxiliary clock is currently enabled, the clock is disabled and then re-enabled with the new rate.

Returns OK or ERROR.

See Also sysLib, sysAuxClkEnable(), sysAuxClkRateGet()

sysBspRev(2)

Name sysBspRev() - return the bsp version with the revision e.g. 1.0/<x>

Synopsis
```
char * sysBspRev (void)
```

Description This function returns a pointer to a bsp version with the revision e.g. 1.0/<x>.
BSP_REV defined in config.h is concatenated to BSP_VERSION defined in bspVersion.h and returned.

Returns A pointer to the BSP version/revision string.

See Also sysLib

sysBusIntAck(2)

Name sysBusIntAck() - acknowledge a bus interrupt

Synopsis

```
int sysBusIntAck
    (
    int  intLevel  /* interrupt level to acknowledge */
    )
```

Description This routine acknowledges a specified VMEbus interrupt level. The VIC064 performs this function automatically.

Returns NULL. Performed by hardware.

See Also sysLib, sysBusIntGen()

sysBusIntGen(2)

Name sysBusIntGen() - generate a bus interrupt

Synopsis

```
STATUS sysBusIntGen
    (
    int  level,  /* VMEbus interrupt level to generate
                    (1-7) */
    int  vector  /* interrupt vector to generate (0-255) */
    )
```

Description This routine generates a bus interrupt for a specified level with a specified vector.

Returns OK, ERROR if interrupt level out of range (1 - 7) or vector is out of range (0 - 255) or a previous unacknowledged interrupt is pending.

See Also sysLib, sysBusIntAck()

sysBusTas(2)

Name sysBusTas() - test and set a location across the bus

Synopsis

```
BOOL sysBusTas
    (
    char  *adrs  /* address to be tested and set */
    )
```

Description This routine performs a 680x0 test-and-set instruction across the backplane.

Returns TRUE (successful set), or FALSE (failure).

See Also sysLib, vxTas()

sysBusToLocalAdrs(2)

Name sysBusToLocalAdrs() - convert a bus address to a local address

Synopsis

```
STATUS sysBusToLocalAdrs
    (
    int   adrsSpace,    /* bus address space in which busAdrs
                           resides */
    char  *busAdrs,    /* bus address to convert */
    char  **pLocalAdrs /* where to return local address */
    )
```

Description This routine gets the local address that accesses a specified VMEbus address.

Returns OK, or ERROR if the address space is unknown or the mapping is not possible.

See Also sysLib, sysLocalToBusAdrs()

sysClkConnect(2)

Name sysClkConnect() - connect a routine to the system clock interrupt

Synopsis

```
STATUS sysClkConnect
    (
    FUNCPTR  routine,  /* routine called at each system clock
                          interrupt*/
    int      arg       /* argument with which to call
                          routine */
    )
```

Description This routine specifies the interrupt service routine to be called at each clock interrupt. It is called from usrRoot() in usrConfig.c to connect usrClock() to the system clock interrupt. It also connects the clock error interrupt service routine.

Returns OK, or ERROR if the routine cannot be connected to the interrupt.

See Also sysLib, intConnect(), usrClock(), sysClkEnable()

sysClkDisable(2)

Name sysClkDisable() - turn off system clock interrupts

Synopsis `void sysClkDisable (void)`

Description This routine disables system clock interrupts.

Returns N/A

See Also sysLib, sysClkEnable()

sysClkEnable(2)

Name sysClkEnable() - turn on system clock interrupts

Synopsis
```
void sysClkEnable (void)
```

Description This routine enables system clock interrupts.

Returns N/A

See Also sysLib, sysClkConnect(), sysClkDisable(), sysClkRateSet()

sysClkInt(2)

Name sysClkInt() - handle a system clock interrupt

Synopsis
```
void sysClkInt (void)
```

Description This routine handles a system clock interrupt. It acknowledges the interrupt and calls the routine installed by sysClkConnect().

See Also sysLib

sysClkRateGet(2)

Name sysClkRateGet() - get the system clock rate

Synopsis `int sysClkRateGet (void)`

Description This routine returns the interrupt rate of the system clock.

Returns The number of ticks per second of the system clock.

See Also sysLib, sysClkEnable(), sysClkRateSet()

sysClkRateSet(2)

Name sysClkRateSet() - set the system clock rate

Synopsis

```
STATUS sysClkRateSet
    (
    int  ticksPerSecond   /* number of clock interrupts per
                             second */
    )
```

Description This routine sets the interrupt rate of the system clock. The new interrupt rate is saved. If the system clock is currently enabled, the clock is disabled and then re-enabled with the new rate. This routine is called by usrRoot() in usrConfig.c.

Note The valid range for the system clock is 40 to 5000 ticks per second.

Returns OK, or ERROR if the tick rate is invalid or the timer cannot be set.

See Also sysLib, sysClkEnable(), sysClkRateGet()

sysClockGet(2)

Name sysClockGet() - get the contents of the clock

Synopsis

```
time_t sysClockGet
    (
    struct tm  *time
    )
```

Description This routine copies the time from the time keeper device into the time structure provided as the input argument (see time.h). The function will automatically calculate the day of the week from the time provided in the time keeper device.

Returns ERROR, if time keeper is not running; otherwise the calendar time.

See Also sysLib, sysClockSet(), sysClockStop()

sysClockPrint(2)

Name sysClockPrint() - prints value stored in the nonvolatile clock

Synopsis
```
STATUS sysClockPrint (void)
```

Description This routine prints the specified time as set in the nonvolatile clock.

Returns OK

See Also sysLib, sysClockGet()

sysClockSet(2)

Name sysClockSet() - sets the nonvolatile clock

Synopsis

```
STATUS sysClockSet
    (
    UINT8  date,    /* date (1 - 31)    */
    UINT8  month,   /* month (1 - 12)   */
    UINT8  year,    /* year (0 - 99)    */
    UINT8  hour,    /* hours (0 - 23)   */
    UINT8  minute,  /* minute (0 - 59)  */
    UINT8  second   /* seconds (0 - 59) */
    )
```

Description This routine sets the clock to the specified time. If the clock was stopped, this function will start the clock.

Returns OK

See Also sysLib, sysClockGet(), sysClockStop()

sysClockStop(2)

Name sysClockStop() - stops the nonvolatile clock

Synopsis
```
STATUS sysClockStop (void)
```

Description This routine stops the nonvolatile clock such that the clock battery can be conserved.

Returns OK

See Also sysLib, sysClockSet()

sysFrontPanelSwitch(2)

Name sysFrontPanelSwitch() - read the front panel switches

Synopsis

```
int sysFrontPanelSwitch
    (
    int  swNum  /* switch number to read */
    )
```

Description This routine returns the value of the specified front panel switch.

Returns Value that switch is set to.

See Also sysLib

sysHwInit(2)

Name sysHwInit() - initialize the CPU board hardware

Synopsis
```
void sysHwInit (void)
```

Description This routine initializes various features of the CPU board hardware. It is called from usrInit() in usrConfig.c.

Note This routine should not be called by the user.

Returns N/A

See Also sysLib

sysIlaccIntEnable(2)

Name sysIlaccIntEnable() - enable the ILACC interrupt level

Synopsis

```
int sysIlaccIntEnable
    (
    int  level  /* interrupt level, not used */
    )
```

Description This routine enables interrupts for the on-board ILACC chip at a specified level. ILACC interrupts are controlled by the VIC chip. The VIC chip also provides the interrupt vector.

Returns OK

See Also sysLib

sysIntDisable(2)

Name sysIntDisable() - disable a bus interrupt level

Synopsis

```
STATUS sysIntDisable
    (
    int  intLevel  /* interrupt level to disable (1-7) */
    )
```

Description This routine disables a specified VMEbus interrupt level.

Returns OK, or ERROR if <intLevel> is not in the range 1 - 7.

See Also sysLib, sysIntEnable()

sysIntEnable(2)

Name sysIntEnable() - enable a bus interrupt level

Synopsis

```
STATUS sysIntEnable
    (
    int  intLevel  /* interrupt level to enable (1-7) */
    )
```

Description This routine enables a specified VMEbus interrupt level.

Returns OK, or ERROR if <intLevel> is not in the range 1 - 7.

See Also sysLib, sysIntDisable()

sysLocalToBusAdrs(2)

Name sysLocalToBusAdrs() - convert a local address to a bus address

Synopsis

```
STATUS sysLocalToBusAdrs
    (
    int   adrsSpace,   /* bus address space in which busAdrs
                          resides */
    char  *localAdrs,  /* local address to convert */
    char  **pBusAdrs   /* where to return bus address */
    )
```

Description This routine gets the VMEbus address that accesses a specified local memory address.

Returns OK, or ERROR if the mapping is not possible.

See Also sysLib, sysBusToLocalAdrs()

sysMailboxConnect(2)

Name sysMailboxConnect() - connect a routine to the mailbox interrupt

Synopsis

```
STATUS sysMailboxConnect
    (
    FUNCPTR  routine,  /* routine called at each mailbox
                          interrupt */
    int      arg       /* argument with which to call
                          routine */
    )
```

Description This routine specifies the interrupt service routine to be called at each mailbox interrupt.

Returns OK, always

See Also sysLib, intConnect(), sysMailboxEnable()

sysMailboxEnable(2)

Name sysMailboxEnable() - enable the mailbox interrupt

Synopsis

```
STATUS sysMailboxEnable
    (
    char  *mailboxAdrs  /* address of mailbox (ignored) */
    )
```

Description This routine enables the ICMS-0 mailbox on the VIC064.

Returns OK, always.

See Also sysLib, sysMailboxConnect()

sysMemTop(2)

Name sysMemTop() - get the address of the top of memory

Synopsis
```
char *sysMemTop (void)
```

Description This routine finds the size of memory.

Returns The address of the top of memory.

See Also sysLib

sysModel(2)

Name sysModel() - return the model name of the CPU board

Synopsis
```
char *sysModel (void)
```

Description This routine returns the model name of the CPU board.

Returns A pointer to the string "ELTEC EUROCOM-27".

See Also sysLib

sysNvRamGet(2)

Name sysNvRamGet() - get the contents of nonvolatile RAM

Synopsis

```
STATUS sysNvRamGet
    (
    char  *string,  /* where to copy nonvolatile RAM    */
    int   strLen,   /* maximum number of bytes to copy  */
    int   offset    /* byte offset into nonvolatile RAM */
    )
```

Description This routine copies the contents of nonvolatile memory into a specified string. The string will be terminated with an EOS.

Returns OK, or ERROR if access is outside the nonvolatile RAM range.

See Also sysLib, sysNvRamSet()

sysNvRamSet(2)

Name sysNvRamSet() - write to nonvolatile RAM

Synopsis

```
STATUS sysNvRamSet
    (
    char  *string,  /* string to be copied into nonvolatile
                       RAM */
    int   strLen,   /* maximum number of bytes to copy */
    int   offset    /* byte offset into nonvolatile RAM */
    )
```

Description This routine copies a specified string into nonvolatile RAM.

Returns OK, or ERROR if access is outside the nonvolatile RAM range.

See Also sysLib, sysNvRamGet()

sysProcNumGet(2)

Name sysProcNumGet() - get the processor number

Synopsis
```
int sysProcNumGet (void)
```

Description This routine returns the processor number for the CPU board, which is set with sysProcNumSet().

Returns The processor number for the CPU board.

See Also sysLib, sysProcNumSet()

sysProcNumSet(2)

Name sysProcNumSet() - set the processor number

Synopsis

```
void sysProcNumSet
    (
    int  procNum
    )
```

Description This routine sets the processor number for the CPU board. Processor numbers should be unique on a single backplane.

Returns N/A

See Also sysLib, sysProcNumGet()

sysScsiInit(2)

Name sysScsiInit() - initialize NCR720 SCSI chip

Synopsis
```
STATUS sysScsiInit (void)
```

Description This routine creates and initializes an SIOP structure, enabling use of the on-board SCSI port. It also connects the proper interrupt service routine to the desired vector, and enables the interrupt at the desired level.

Returns OK, or ERROR if the control structure is not created or the interrupt service routine cannot be connected to the interrupt.

See Also sysLib

sysScsiMode(2)

Name sysScsiMode() - change the SCSI mode for a target

Synopsis

```
STATUS sysScsiMode
    (
    int         mode,  /* transfer mode 0 = narrow async */
    1 = narrow  sync
    2 = wide    async
    3 = wide    sync
    */
    int         scid   /* SCSI ID                        */
    )
```

Description This routine changes the operating mode of an SCSI target from narrow to wide or asynchronous to synchronous. This facility should be located in the scsiLib eventually.

Returns OK, or ERROR if the mode change could not be accomplished.

See Also sysLib

sysSetLed(2)

Name sysSetLed() - set value in LED

Synopsis

```
int sysSetLed
    (
    int  hexVal  /* value to display in LED display */
    )
```

Description This routine sets a value into the hex display. If the value to display is negative, the display will be blanked.

Returns OK

See Also sysLib

sysSysfailConnect(2)

Name sysSysfailConnect() - connect a routine to the SYSFAIL signal

Synopsis

```
STATUS sysSysfailConnect
    (
    VOIDFUNCPTR  routine,  /* routine to be connected to
                              SYSFAIL signal */
    int          arg       /* argument with which to call
                              routine */
    )
```

Description This routine connects a specified routine to the board's SYSFAIL signal.

Returns OK, or ERROR if the routine cannot be connected to the interrupt.

See Also sysLib, intConnect()

sysToMonitor(2)

Name sysToMonitor() - transfer control to the ROM monitor

Synopsis

```
STATUS sysToMonitor
    (
    int  startType  /* parameter passed to ROM to tell it
                       how to boot */
    )
```

Description This routine transfers control to the ROM monitor. Normally, it is called only by reboot() -- which services ^X -- and bus errors at interrupt level. However, in some circumstances, the user may wish to introduce a <startType> to enable special boot ROM facilities.

Returns Does not return.

See Also sysLib

sysVicBlkAdj(2)

Name sysVicBlkAdj() - adjust burst length and interleave period for DMA transfers

Synopsis

```
STATUS sysVicBlkAdj
    (
    int  burstLength,  /* burst length of DMA block transfer
                          (1-64) */
    int  interleave    /* interleave period between bursts
                          (1-15) */
    )
```

Description This routine changes the burst length and interleave period that are to be used for VME DMA transfers. <burstLength> is the number of VMEbus cycles per burst, <interleave> is the local cycle interleave period between block transfers. The interleave time is equal to 250 ns times <interleave>.

Returns OK, or ERROR if parameters are out of range.

See Also sysLib, sysVicBlkEnable(), sysVicBlkCopy()

sysVicBlkCopy(2)

Name sysVicBlkCopy() - copy blocks over the VMEbus using the VIC's DMA feature

Synopsis

```
STATUS sysVicBlkCopy
    (
    char *  localAddr,   /* local address */
    char *  remoteAddr,  /* vme address */
    int     nbytes,      /* transfer length in bytes */
    BOOL    toVme,       /* copy local memory to vme
                            (or vice-versa) */
    BOOL    doD64        /* do D64 transfer */
    )
```

Description This routine uses the DMA feature of the VIC to do block-mode copies to/from the VMEbus. This routine takes a local address <localAddr> and remote address <remoteAddr> and transfers <nbytes> bytes in the direction specified by <toVme>.

Returns OK, or ERROR if transfer was unsuccessful. Unsuccessful transfers result from bus errors.

See Also sysLib, sysVicBlkEnable(), sysVicBlkAdj()

sysVicBlkEnable(2)

Name sysVicBlkEnable() - initialize the DMA copy operation for the VIC064.

Synopsis

```
STATUS sysVicBlkEnable
    (
    VOIDFUNCPTR *  vector,  /* interrupt vector for DMA done
                               interrupt */
    int            level,   /* interrupt level for DMA done
                               interrupt */
    int       burstLength,  /* burst length of DMA block
                               transfer  (1-64)*/
    int          interleave /* interleave period between
                               bursts (1-15) */
    )
```

Description This routine initializes operations for DMA transfers using the VIC064. The DMA done interrupt is attached using the <vector> as the interrupt vector and <level> as the interrupt level. <burstLength> is the number of VMEbus cycles per burst, <interleave> is the local cycle interleave period between block transfers. The interleave time is equal to 250 ns times <interleave>.

This routine is typically called by sysHwInit2().

Returns OK or ERROR, if the DMA parameters are not valid.

See Also sysLib, sysVicBlkAdj(), sysVicBlkCopy()

sysVicShow(2)

Name sysVicShow() - displays the contents of the VIC064 registers

Synopsis `int sysVicShow()`

Description This routine displays the contents of the VIC064 registers.

Returns OK, always.

See Also sysLib,

tyCoDevCreate(2)

Name tyCoDevCreate() - create a device for an on-board serial port

Synopsis

```
STATUS tyCoDevCreate
    (
    char  *name,      /* name to use for this device      */
    int   channel,    /* physical channel for this device */
    int   rdBufSize,  /* read buffer size, in bytes       */
    int   wrtBufSize  /* write buffer size, in bytes      */
    )
```

Description This routine creates a device for a specified tty port. Each port to be used should have exactly one device associated with it by calling this routine. When the graphics/keyboard device is included, it is considered to be channel 0 and the serial ports are considered to be channels 1 through 4. When the graphics/keyboard device is not included in the VxWorks kernel then the serial ports are considered to be channels 0 through 3.

For instance, to create the device "/tyCo/0", with buffer sizes of 512 bytes, the proper call would be:

```
tyCoDevCreate ("/tyCo/0", 0, 512, 512);
```

Returns OK, or ERROR if the driver is not installed, the channel is invalid, or the device already exists.

See Also tyCoDrv, tyCoDrv()

tyCoDrv(2)

Name tyCoDrv() - initialize the tty driver

Synopsis

```
STATUS tyCoDrv (void)
```

Description This routine initializes the tty driver for the ELTEC EUROCOM-27. It sets up the serial devices and the graphics/keyboard device.

The graphics/keyboard device may optionally be omitted from the VxWorks kernel by undefining the `INCLUDE_GRAPHICS_KEYBOARD`.

This routine will initialize the driver and the hardware for the serial ports and the graphics/keyboard device.

This routine should be called exactly once, before any reads, writes, or calls to tyCoDevCreate(). Normally, it is called by usrRoot() in usrConfig.c.

Returns OK or ERROR if the driver cannot be installed.

See Also tyCoDrv, tyCoDevCreate()